Announcements

- Midterm exam
  - Thursday, February 13
  - One hour, In class, Closed book
  - Exercises at the end of the chapters are good to study
  - No code writing, but you have to read and understand Java code
Announcements

- Chapters 1 - 7 in Java By Dissection
  - Especially classes and inheritance in chapters 5 - 7
- Study concepts and algorithms needed for the first three assignments.
- Review Session
  - Wednesday, Feb. 12, evening, somewhere in Baskin
Advanced Swing Programming
CMPS 109 - Advanced Programming
February 6, 2003
Custom Components

- Two major branch points for custom components:
  - JPanel
  - JComponent
- What kind of component are you building?
Composite Components

- JPanel
  - if you intend to create a component that combines other, existing swing components.

- Example
  - A Spinner is a text field that allows the user to enter a range of integers with an up arrow and down arrow on the right to scroll through values.
public class Spinner
    extends JPanel
{
    private JTextField textField;
    private JButton upButton;
    private JButton downButton;

    private int currentValue;

    public int getValue() { return( currentValue ); }
    public void setValue( int newValue )
    {
        currentValue = newValue;
        textField.setText( Integer.toString( currentValue ) );
    }
public Spinner()
{
    textField = new JTextField( Integer.toString( currentValue ) );
    textField.setMaximumSize( new Dimension(
        textField.getMaximumSize().width,
        textField.getPreferredSize().height ) );
    textField.addActionListener( new TextFieldListener() );

    upButton = new JButton( "Up" );
    upButton.addActionListener( new UpButtonListener() );

    downButton = new JButton( "Down " );
    downButton.addActionListener( new DownButtonListener() );

    setLayout( new BoxLayout( this, BoxLayout.X_AXIS ) );
    add( textField );
    add( Box.createHorizontalStrut( 4 ) );
    add( upButton );
    add( Box.createHorizontalStrut( 4 ) );
    add( downButton );
    addBorder( BorderFactory.createEtchedBorder() );
}

Spinner Example
private class UpButtonListener implements ActionListener {
    public void actionPerformed( ActionEvent e ) {
        setValue( getValue() + 10 );
    }
}

private class DownButtonListener implements ActionListener {
    public void actionPerformed( ActionEvent e ) {
        setValue( getValue() - 10 );
    }
}

private class TextFieldListener implements ActionListener {
    public void actionPerformed( ActionEvent e ) {
        try {
            setValue( Integer.parseInt( textField.getText() ) );
        } catch( NumberFormatException exception ) {} 
        textField.setText( Integer.toString( getValue() ) );
    }
}

Spinner Example
New Components

- Sometimes you want to create entirely new components.
- Speedometer, Chart, Plot, Graph
- If the new components are not types of existing Swing components, subclass JComponent.
JComponent

- JComponent is the base class of almost every Swing component.
  - Allows “pluggable” look and feel.
  - Supports keyboard handling.
  - Supports ToolTips and accessibility
  - Supports component properties
  - Supports painting, double buffering, and borders.
Building a Component

- We will build a simple drawing pad.
- What data is needed?
  - Pen color
  - Pen size
  - A place to draw.
  - The current size (width and height).
public class DrawPad
    extends JComponent
{
    public static final float MAX_PEN_SIZE = 10.0f;
    public static final float MIN_PEN_SIZE = 1.0f;

    private Image canvas;

    private int currentWidth;
    private int currentHeight;

    private Color currentColor;
    private float currentSize;
Initial Configuration

- JComponents are invisible by default.
- The default sizes are all 0.
- Initial settings for the pen.
public DrawPad()
{
    setOpaque( true );
    setBackground( Color.white );

    currentWidth = 320;
    currentHeight = 240;

    currentColor = Color.black;
    currentSize = MIN_PEN_SIZE;

    setMinimumSize( new Dimension( 25, 25 ) );
    setPreferredSize( new Dimension( currentWidth, currentHeight ) );

    addMouseMotionListener( new PadMouseMotionListener() );
}
Accessors

- We need to get and set the size of the pen.
- We also need to get and set the color of the pen.
DrawPad Accessors

public void setPenColor( Color newColor )
{
    if( newColor != currentColor )
    {
        currentColor = newColor;
    }
}

public Color getPenColor()
{
    return( currentColor );
}

public void setPenSize( float newSize )
{
    if( ( newSize != currentSize ) &&
        ( newSize >= MIN_PEN_SIZE ) && ( newSize <= MAX_PEN_SIZE ) )
    {
        currentSize = newSize;
    }
}

public float getPenSize()
{
    return( currentSize );
}
Moving the Mouse

- This is a drawing pad, so the movement of the mouse needs to be tracked.
- Always know where the mouse’s last known position is.
- Draw when the mouse button is pushed.
- Draw into an offscreen buffer.
Offscreen Buffers

- You can create instances of the Image class to store the current picture displayed in the component.
- The component must be visible first.
- The Image instance has a Graphics instance to handle drawing.
Using the Image class

- Use the function:
  - Image createImage( int width, int height );

- You can place Images into JLabels, JButtons, JTrees, and other classes
  - Pass the Image to a constructor of the ImageIcon class when you need to create an Icon.
Graphics

- First there was Graphics
  - Able to draw: Arcs, Text, Images, Lines, Ovals, Polygons, Rectangles, Round Rectangles
  - Filled or not
  - Change the color or font
  - Set a clip region
    - Clip regions are limiting areas that can prevent you from “drawing outside the lines”. 
And then there was Graphics2D

- All Graphics instances are actually Graphics2D
- Provides advanced text and rendering support.
- Generic Shapes and Strokes
- Transformations: rotate, scale, shear, translate
- Anti-aliasing
- “Hit” testing for shapes.
Graphics2D

- Examples of shapes, fills, and strokes
- Source code is available at:

  http://java.sun.com/docs/books/tutorial/2d/display/strokeandfill.html
private class PadMouseMotionListener extends MouseMotionAdapter {
    private int lastX;
    private int lastY;

    public void mouseMoved( MouseEvent e ) {
        lastX = e.getX();
        lastY = e.getY();
    }

    public void mouseDragged( MouseEvent e ) {
        if( canvas == null ) {
            updateCanvasSize();
        }
        Graphics2D g = (Graphics2D) canvas.getGraphics();
        g.setColor( currentColor );
        g.setStroke( new BasicStroke( currentSize, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND ) );
        g.drawLine( lastX, lastY, e.getX(), e.getY() );
        lastX = e.getX();
        lastY = e.getY();
        repaint();
    }
}
Painting a component

- JComponent is a subclass of Component in java.awt.
- It has four paint related functions:
  - paint( Graphics g )
  - paintBorder( Graphics g )
  - paintChildren( Graphics g )
  - paintComponent( Graphics g )
Painting a component

- Standard practice in Swing dictates that you override `paintComponent(Graphics g)` to perform custom painting.
- Calling `repaint()` on the component will trigger an update at the end of the current event loop.
public void paintComponent( Graphics g )
{
    super.paintComponent( g );
    g.fillRect( 0, 0, currentWidth, currentHeight );
    g.drawImage( canvas, 0, 0, null );
}

private void updateCanvasSize()
{
    Image oldCanvas = canvas;

    canvas = createImage( currentWidth, currentHeight );

    Graphics2D g = (Graphics2D) canvas.getGraphics();

    g.setColor( getBackground() );
    g.fillRect( 0, 0, currentWidth, currentHeight );

    if( oldCanvas != null )
    {
        // Copy old canvas to new one.
        g.drawImage( oldCanvas, 0, 0, null );
    }
}
Resizing and Bounds

- Layout Managers control how big your component actually is.
- Some layout managers ignore the minimum, preferred, and maximum sizes.
- Override the function `setBounds(int x, int y, int width, int height)`.
public void setBounds( int x, int y, int width, int height )
{
    currentWidth = width;
    currentHeight = height;
    updateCanvasSize();

    super.setBounds( x, y, width, height );
    repaint();
}
Model View Controller

- Model-View-Controller (MVC) is a classic method of designing the architecture for a GUI.
- Many of the Swing classes use MVC.
Model

- The model is a class that contains the data being displayed.
- Accessor functions allow the data to be changed.
- The model notifies the controller when changes are made.
View

- The view is responsible for displaying the data.
- One Model can drive multiple views and some views use more than one Model.
- The view notifies the controller when events occur.
Controller

- The glue between the Model and the View.
- When the model changes, the Controller notifies the view.
- The Controller handles events and updates the model as appropriate.
How MVC can help you

- Combine the View and Controller into one class
  - This would be the JComponent subclass that draws your GraphView
  - Not much Controller functionality required
    - Updates from the Model
- The Model is the same old Graph class
Your GUI - Step by Step

1. Draw your interface on paper
2. Make a JFrame and add pre-built Swing components (buttons, labels, etc.)
3. Build the data model class
4. Handle events and update the Model
5. Fill in the blanks in the GraphView.
A Simple Graph Display
Other Swing Features

- ToolTips
- Loading Images
- Scrolling
  - Don’t shrink the image, just add scroll bars.
- JList
  - An example of a complex Swing component.
**ToolTip**

- A ToolTip is (usually) short, descriptive text that helps the user understand what a component does.
- JComponent provides the function `setToolTipText(String text);`
- ToolTips can be changed at any time.
Loading Images

- Before you can use an image in your program, you have to load it.
  - For simplicity, keep an image in the same directory as the class that will use it.

- URL Class.getResource(String name);
  - new Image(this.getClass().getResource("BtnPic.jpg"));

- Using HTML tags should also work.
Scrolling

- JScrollPane does almost all of the work
  - new JScrollPane( someBigComponent );
- The component being scrolled should implement the Scrollable interface.
  - The component’s size
  - Block and Unit increments
  - Forcing the height or width (like auto-wrap text)
ScrollView

Dimension `getPreferredSize()`
Returns the preferred size of the viewport for a view component.

```java
int getScrollableBlockIncrement (Rectangle visibleRect, int orientation, int direction)
```
Components that display logical rows or columns should compute the scroll increment that will completely expose one block of rows or columns, depending on the value of orientation.

```java
boolean getScrollableTracksViewportHeight ()
```
Return true if a viewport should always force the height of this Scrollable to match the height of the viewport.

```java
boolean getScrollableTracksViewportWidth ()
```
Return true if a viewport should always force the width of this Scrollable to match the width of the viewport.

```java
int getScrollableUnitIncrement (Rectangle visibleRect, int orientation, int direction)
```
Components that display logical rows or columns should compute the scroll increment that will completely expose one new row or column, depending on the value of orientation.
JList

- JList provides the view of a list of items
The items in the list are stored in a class that implements the ListModel interface.

JList adds Listeners to the model so it will be notified of changes.

The model provides the number of items and indexed access, like an array.
ListSelectionModel

- The JList allows different ways to deal with selections through a class that implements the ListSelectionModel.
  - It’s a huge interface.
- The ListSelectionModel keeps track of which items are selected by their index.
ListCellRenderer

- Lists can display anything
- A class implementing the interface `ListCellRenderer` is handles drawing
- Each cell’s data is passed to the renderer, one at a time.
Default Implementations

- DefaultListCellRenderer
  - Handles the common data types placed in lists
  - Text and Images

- DefaultListSelectionModel
  - Supports single selection and multiple selection

- DefaultListModel
  - Provides functions to maintain the list data.
Models Change Often

- `JList.setModel()` changes the data model
- `JList.setSelectionModel()` changes the selection model
- `JList.setCellRenderer()` changes the renderer
Why change?

- Sort the items in different ways
  - Alphabetical, reverse alphabetical, size, color, etc.
- You want to display a custom component in the list
- There are combinations of items that cannot be selected together.
Listener Lists

- All these models have to keep track of listeners
- The listeners are notified when changes are made.
- Swing provides a helper class called EventListenerList.
EventListenerList

- `javax.swing.event.EventListenerList`
- Add and remove listeners.
- Fast access when event notifications are sent
- JavaDoc page provides nearly complete pseudo-code for it (just fill in your event)
Questions
and
Demonstrations